Future of Software Engineering

Kevin Sullivan
University of Virginia

Why is it Hard?

- Demands creativity, knowledge, structure
- Very few natural, order-creating constraints
- Hard to reason about complex logical structures
- Complex, multi-objective, human fitness functions
- Incomplete information: design, environment, fitness
- Decentralized evolution of design and environment
- Delayed discovery of weaknesses, adverse interactions
- Complex connection of design structure to evolvability
- *SE cookbooky/heuristic: what is science of SW design?*

What's Currently Infeasible?

- Engineering SW risk-return characteristics
- Engineering of bio-scale software systems
  - "…the limits of software engineering have been clear now for two decades. The biggest programs anyone can build are about ten million lines of code. A real biological object — a creature, an ecosystem, a brain — is something with the same complexity as ten billion lines of code. And how do we get there?" –Jordan Pollack
  - E.g., 10 x 10 x 10 x 10 design hierarchy of powerpoint-scale modules
- Reasoning about critical, specified properties

Radical Directions

- Strategic software design: *value-based science of design*
  - Economics (utility, capital market value—options value of modularity)
  - Biology/CAS (evolution on fitness landscapes—parameter-based design)
  - Social Sciences (cognitive costs, sociology—e.g., participatory design)
  - Humanities? (aesthetic, cultural, historical, ethical measures of value)
- Layered, property-oriented *design of design rules*
  - Among other things, necessary for systematic COTS integration—e.g., POP
  - *Terrific* target for use of formal methods—e.g., design of COM
  - Beyond connector-component ontology for foundational software design
  - Analog: rules of physics, then chemistry, then biology, then ecology, …
- Lightweight architectural aspects for *emerging noosphere*
  - Anticipating "software in everything"
  - How to understand, track, manage vastly more complex software
  - E.g., arbitrary running objects expose web interfaces

Challenge

- Understand the conditions necessary to transform the software industry to one that looks more like the PC industry: firms compete over standardized components that can be integrated into systems with specified cost and performance properties
- Reevaluate design of the emerging global grid from ground up from a property-based perspective including focus on dependability characteristics (bandwidth/throughput getting cheap, now what?)

Designing an R&D Portfolio

- Emphasize need for *theory of software design* having both intellectual depth and descriptive & prescriptive potential (not just the scientific method applied to the testing of ad hoc ideas)
- Increase emphasis on *intellectually clear & compelling advances* (e.g., bio-scale software, breakthrough models of modularity & evolution, restructuring SW industry...)
- Treat R&D as an investment activity: projects are a portfolio of options (to abandon, expand in phases); *requires dynamic investment management approach*; need to coordinate some to use options most effectively